VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Matúš Ozaniak

Bakalářská práce

Vedoucí práce: Ing. Pavel Dohnálek, Ph.D.

Ostrava, 2021

Abstrakt

V tejto bakalárskej práci sa venujem opisu priebehu mojej odbornej praxe v spoločnosti profiq s.r.o.. Mojím zadaním na odbornej praxi bolo implementovať prvky grafického rozhrania podľa vopred pripravených dizajnových návrhov pre platformu na organizáciu podujatí a ich logistiku. Práca sa zaoberá predstavením spoločnosti profiq a môjho pracovného zaradenia. V ďalšej časti opíšem platformu Planesty a použité technológie. Následne opíšem moju implementáciu vybraných grafických komponent.

Kľúčové slová

React-JS; Typescript; grafické rozhranie; bakalárska práca

Abstract

In this bachelor thesis I will describe my individual professional practice in the profiq s.r.o. company. My task in professional practice was to implement graphical user interface elements according to pre-prepared design proposals for platform used to organize events and their logistics. The work introduces the company and my job classification. In the next part I will describe the Planesty platform and the technologies I used. Then I will describe my implementation of selected graphical components.

Keywords

React-JS; Typescript; graphical user interface; master thesis

Poďakovanie

Rád by som na tomto mieste poďakoval Mgr. Gabrielovi Puhallovi za umožnenie absolvovania bakalárskej praxe vo firme, vedúcemu tejto práce pánovi Ing. Pavlovi Dohnálkovi, Ph.D. a spolupracovníkom vo firme.

Obsah

Zoznam použitých skratiek a symbolov

CSV	–	Comma Separated Values
BEO	–	Banquet Event Order
JS	–	JavaScript
ID	–	Identifikátor
API	–	Application Programming Interface

Zoznam obrázkov

Kapitola 1

Úvod

Svojú odbornú bakalársku prax som absolvoval v spoločnosti profiq s.r.o. [1], kde som pracoval na zákazkovej aplikácií Planesty [2]. V tejto práci popisujem, ako som sa zoznámil s projektom a ako som následne implementoval na projekte nové funkcionality.

Po zoznámení sa s projektom a používanými technológiami, bolo mojou úlohou implementovať komponenty grafického rozhrania na frontendovej časti aplikácie, určenej na organizáciu podujatí. Frontendová časť aplikácie bude slúžiť ako grafické rozrhanie na prezentačnej vrstve aplikácie a bude naprogramovaná v React JS [3] s použitím statického typovania z Typescriptu [4]. Na získavanie dát z aplikačnej vrstvy - backendu, budú použité GraphQL [5] volania na backend API.

Každý komponent mal vopred pripravený grafický návrh, podľa ktorého sa mal implementovať jeho dizajn. Ku komponentom, ktoré potrebovali prístup k dátam z backendu, bola taktiež vopred pripravená API na backende. Vytvorené komponenty budú slúžiť používateľovi na vytváranie a správu udalostí.

1.1 Popis zamerania spoločnosti, v ktorej študent vykonal odbornú prax

Spoločnosť profiq s.r.o. ma hlavné sídlo v Prahe, no má viacero pobočiek v Českej republike a zahraničí. V súčasnej dobe zamestnáva zhruba 60 zamestnancov, z ktorých je väčšina na pozícii programátora. Zaoberá sa outsourcingom služieb v oblasti softverového inžinierstva a vývoja aplikácií pre rôznych zákazníkov z celého sveta. Väčšinu zákazníkov však tvoria technologické spoločnosti zo Spojených štátov amerických.

Spoločnosť poskytuje služby od návrhu a vývoja až po testovanie a údržbu softvéru. Pri tvorbe softvéru sa vo firme väčšinou používa agilný vývoj, vďaka ktorému vývojársky tím rýchlo reaguje na potreby zákazníka. To zahŕňa pravidelné a väčšinou každodenné mítingy celého tímu vývojárov pracujúcich na určitom projekte. Na týchto mítingoch tím konzultuje požiadavky zákazníka a spôsoby, či poradie ich riešenia.

Firma sa orientuje najmä na vývoj webových a mobilných aplikácií, cloud a bezpečnosť. Medzi hlavných zákazníkov firmy profiq patria napríklad Divvy, ktoré ponúka platformu na správu výdajov pre korporátne firmy alebo ForgeRock, ktorý poskytuje aplikáciu na správu identít. Medzi dalších známych zákazníkov patrí Avast, T-mobile alebo O_2.

Obr. 1.1: Logo profiq s.r.o.

1.2 Moje pracovné zaradenie

Majitelia platformy Planesty potrebovali lepšie fungujúcu a modernejšie vyzerajúcu aplikáciu. Predchádzajúca verzia aplikácie používala zastaralý dizajn grafického rozhrania a trpela výkonnostnými problémami. Okrem toho v starej verzii neboli vyvinuté všetky funkcionality v rozsahu, v ktorom by si majitelia priali, a tak sa rozhodli nechať novú platformu navrhnúť a vyvinúť firmou profiq.

Firma vytvorila tím vývojárov na projekt zaoberajúci sa vývojom platformy Planesty. Do tímu im ešte chýbal vývojár, ktorý by mal na starosti vývoj nových komponentov grafického rozhrania, a tak ma v rámci bakalárskej praxe zaradili do tímu. V tíme som plnil úlohu junior frontend vývojára a mojou prácou bola implementácia, a oprava chýb frontendových komponentov aplikácie Planesty.

V rámci bakalárskej praxe mi bolo priradených niekoľko úloh týkajúcich sa implementácie nových funkcionalít grafického rozhrania do aplikácie.

Kapitola 2

Predstavenie platformy Planesty

Platforma Planesty [2] slúži na organizáciu podujatí a logistiku s tým spojenú. Zákazník môže vytvárať na platforme podujatia spolu so zoznamom úloh a objednávok týkajúcich sa udalosti.

Obr. 2.1: Logo Planesty

2.1 Workspace a podujatia

Zákazník Planesty si predplatí workspace, pod ktorým môže vytvárať podujatia. Počet podujatí - eventov[1], ktoré môže za rok vytvoriť, je určený stupňom predplatného. Zákazník môže pridať do workspace osoby, ktorým potom príde na email pozvánka do daného workspace.

Zákazník pri vytváraní podujatia vyplní potrebné údaje vo formulári na vytvorenie eventu. K podujatiu je možno pridať miesto konania v prípade, ak sa nejedná o virtuálne podujatie. Ďalej je možno pridať stručný popis podujatia, prípadne zoznam štítkov - labelov a logo podujatia. Okrem toho môže zákazník určiť, či sa jedná o súkromné alebo verejné podujatie.

Pre príklad môžeme uviesť Vysokú Školu Banskú: Kompetentná osoba z VŠB si predplatí workspace na platforme a nazve ho *VŠB Workspace*. Pod týmto workspace následne vytvorí podujatie na veľtrh Kariéra PLUS. Vyplní dátum a čas začiatku a konca podujatia, miesto konania - *Nová Aula VŠB* a pridá stručný popis.

[1]Event = podujatie

2.2 Úlohy podujatia

Ku každému podujatiu je možné pridať úlohy a dané úlohy prideliť osobám, ktoré sa majú o úlohy postarať. Osoba, ktorá ma pridelené úlohy potom vidí zoznam týchto úloh spolu s ich prioritou a dátum odbavenia v zozname úloh.

Úlohy sa vytvárajú cez formulár na pridanie úlohy, kde používateľ vyberie akého podujatia sa úloha týka, potom zadá názov úlohy spolu s prioritou odbavenia danej úlohy. Následne pridá osobu zodpovednú za danú úlohu a prípadne aj ďalšie osoby. Vyplní dátum a čas do kedy sa ma úloha splniť a môže vyplniť podrobný popis úlohy. K úlohe je taktiež možné pridať štítky.

Vrátime sa k príkladu z VŠB. K podujatiu Kariéra PLUS môžeme vytvoriť úlohu na pripravenie auly. Úlohe nastavíme dátum a čas pred začiatkom veľtrhu a do popisu pridáme veci, ktoré treba pred veľtrhom pripraviť. Napríklad rozložiť stánky, polepiť informačné letáky atď.

2.3 Objednávka služieb - BEO

Zákazník si môže ku každému podujatiu pridať objednávku služieb - BEO. V objednávke je možné špecifikovať čas a dátum daného zhromaždenia spolu s miestom konania. Ku objednávke je možné pridať zoznam jedál a nápojov, ktoré bude treba zabezpečiť na zhromaždenie. Okrem toho môže používateľ pridať aj zoznam audiovizuálnych pomôcok potrebných ku zhromaždeniu.

K podujatiu Kariéra PLUS pridáme objednávku služieb na zahájenie veľtrhu. Do objednávky pridáme čas a dátum, plus miesto konania. V mieste konania je možné určiť aj konkrétnu miestnosť zhromaždenia. Pridáme zoznam občerstvenia pre návštevníkov, napríklad balené vody na pitie. Ďalej pridáme položky do zoznamu audiovizuálnych pomôcok, napríklad pri zahájení budeme potrebovať zopár mikrofónov a ozvučenie celej miestnosti. Ku každej položke zo zoznamu občerstvenia alebo audiovizuálnych pomôcok je možné zadať počet daných položiek a jednotkovú cenu za položku. Ceny za položky sa následne sčítajú a zobrazia na detaile objednávky.

2.4 Zoznam úloh zadaných študentovi v priebehu odbornej praxe a vyjadrenie ich časovej náročnosti

Súčasťou implementácie budú 3 komponenty obsahujúce formuláre na vytvorenie udalosti, vytvorenie úlohy k udalosti a formulár na vytvorenie objednávky služieb.

Ďalej bude súčasťou implementácie bočné menu na filtrovanie objednávok, menu na import CSV súboru s objednávkami služieb a menu editácie viditeľnosti vybraných informačných prvkov v objednávke.

2.4.1 Zoznámenie sa s použitými technológiami

Prvá úloha bolo naštudovať si, ako sa pracuje s React JS spolu s Typescriptom. Študovanie bolo realizované prostredníctvom niekoľkých video tutoriálov a čítaním dokumentácie daných technológií. Po naštudovaní potrebných materiálov som si mal ešte prejsť dokumentáciu technológie GraphQL a dokumentáciu Material UI [6]. Na naštudovanie technológií mi boli vyhradené 3 dni.

2.4.2 Zoznámenie sa s projektom

Mojou úlohou na celý ďalší týždeň bolo zoznámiť sa projektom, na ktorom budem pracovať. To prebiehalo tak, že mi moji noví kolegovia ukázali základnú orientáciu v platforme Planesty a následne ma nechali zoznámiť sa s funkcionalitami naprieč celou aplikáciou. Zároveň mi dali zoznam funkcionalít, ktoré mám na grafickom rozhraní otestovať. Išlo najmä o zadávanie chybných vstupov do formulárov alebo nahrávanie veľkých súborov. Vďaka tomu som čiastočne pochopil, čo sa snaží aplikácia riešiť a ako funguje. Neskôr mi kolegovia ukázali zdrojový kód aplikácie a dali mi za úlohu pozrieť si, ako sú implementované grafické komponenty. Okrem toho mi ukázali grafické návrhy, zoznam usmernení týkajúcich sa formátovania zdrojového kódu a pracovný tok vývoja nových funkcionalít aplikácie. Od git commitu až po nasadenie na server cez technológiu CI/CD [7]. Grafické návrhy som si prešiel snímok po snímku a tým som dostal predbežnú predstavu o tom, ako ma výsledné grafické rozhranie vyzerať.

2.4.3 Implementácia funkcionalít

V práci mi pridelili zoznam formulárov, ktoré mám za úlohu implementovať. K úlohe mi ukázali, aké GraphQL dotazy na získavanie a ukladanie dát mám používať, a kde sa majú dané komponenty využívať. Na implementáciu každého formulára som mal vyhradený jeden týždeň. Výnimkou bol formulár vytvorenia objednávky, na ktorý som mal dva týždne.

Po dokončení všetkých zadaných formulárov bolo mojou ďalšou úlohou implementovať menu, určené na filtrovanie položiek v zozname. K úlohe mi boli poskytnuté parametre, podľa ktorých mala byť možnosť filtrovať položky. Tieto parametre predstavovali názvy stĺpcov v zozname položiek. Následne bolo mojou úlohou implementovať menu, slúžiace na zapínanie a vypínanie viditeľnosti informačných prvkov pri detaile objednávky. Na tieto dve úlohy mi boli vyhradené tri týždne.

Mojou poslednou úlohou bolo implementovať menu na import CSV súboru. CSV súbor obsahoval zoznam objednávok. Menu malo poskytovať základne informácie k importu a zoznam potrebných parametrov, ktoré musí CSV súbor obsahovať na to, aby bol validný. Na implementáciu importovacieho menu mi bol vyhradený týždeň.

Kapitola 3

Použité technológie

Na projekte boli používané moderné a populárne technológie, čo uľahčovalo vývoj vďaka dobre spracovaným dokumentáciám daných technológií. Vďaka veľkej popularite bolo možné dohľadať na internetových fórach riešenia akýchkoľvek väčších problémov.

3.1 React JS

React alebo React JS je open-source knižnica pre JavaScript [8], určená na tvorbu moderných používateľských rozhraní alebo grafických komponentov. Bola vytvorená Facebookom [9] a je ním aj naďalej spravovaná, spolu s komunitou nezávislých vývojárov.

React sa vyznačuje komponentovým prístupom ku vytváraniu rozhrania. Rozhranie je teda zložené z viacerých menších komponentov. Každý komponent si osobitne spravuje svoju funkcionalitu a vykresľuje svoj obsah. To nám pri vyvíjaní aplikácie Planesty umožnilo vytvárať spoločné komponenty, ktoré sa používali na viacerých miestach súčasne.

Ďalšou zaujímavosťou Reactu je využívanie Virtual DOM, ktorý si vytvára virtuálnu verziu štruktúry webovej stránky a keď sa zmení hodnota nejakých dát, tak porovnáva virtuálnu štruktúru s reálnou štruktúrou web stránky. Ak pri porovnávaní narazí na nejaký rozdiel, tak aktualizuje danú časť web stránky, čím sa zobrazia nové, aktualizované dáta. Vďaka tomu nie je potrebné prekresľovať celú web stránku pri každej zmene dát, čo poskytuje veľké zvýšenie výkonu web stránky. V aplikácií Planesty bolo potreba dynamicky zobrazovať dáta na webovom rozhraní bez toho aby sa musela celá stránka obnovovať a prekresľovať. Okrem toho bolo potrebné, aby sa komponenty na používateľskom rozhraní rýchlo načítavali a vykresľovali, aby používateľ nebol odradený dlhým čakaním na výsledok. Pre tieto aspekty a veľkú popularitu Reactu bola vybraná táto technológia.

3.2 GraphQL

GraphQL je open-source dotazovací jazyk, používaný na tvorbu API. Na rozdiel od najbežnejšie používanej technológie REST [10], ma veľkú výhodu v tom, že v GraphQL nemusíme z API načítavať všetky dáta, ale iba tie dáta alebo parametre, ktoré špecifikujeme v GraphQL dotaze. Vyhneme sa tak zbytočnému načítavaniu dát, ktoré nepotrebujeme. Tým sa zrýchľuje prenos dát a celkový chod aplikácie. Okrem toho musia byť entity používané v GraphQL silno typované. Každá entita tak musí mať nadefinovaný typ, čím sa zamedzí posielaniu nesprávnych dát cez API.

3.3 Typescript

Typescript rozširuje JavaScript o statické typovanie. Typescript nám umožnil otypovať premenné a interfacy, čím sa zamedzí nastavovanie nesprávnych dát do premenných alebo predávanie zlých parametrov do grafických komponentov. Vďaka otypovaným interfacom nám vývojárske nástroje dokázali napovedať, aké parametre treba v rôznych komponentoch nastavovať a akého majú byť typu. Pred spustením aplikácie sa vždy vykonala kontrola typov, ktoré v prípade použitia nesprávneho typu ohlásili chybu.

Pri používaní Typescriptu je možné načítať typy entít pomocou GraphQL dotazu, čo uľahčilo prácu s prijatými dátami z backendu. Namiesto vytvárania tried pre dátové entity stačilo získať typy týchto entít GraphQL dotazom. Kombinácia Typescriptu a GraphQL tak predstavuje obrovskú výhodu oproti klasickému Javascriptu, a preto boli tieto technológie začlenené do projektu.

Kapitola 4

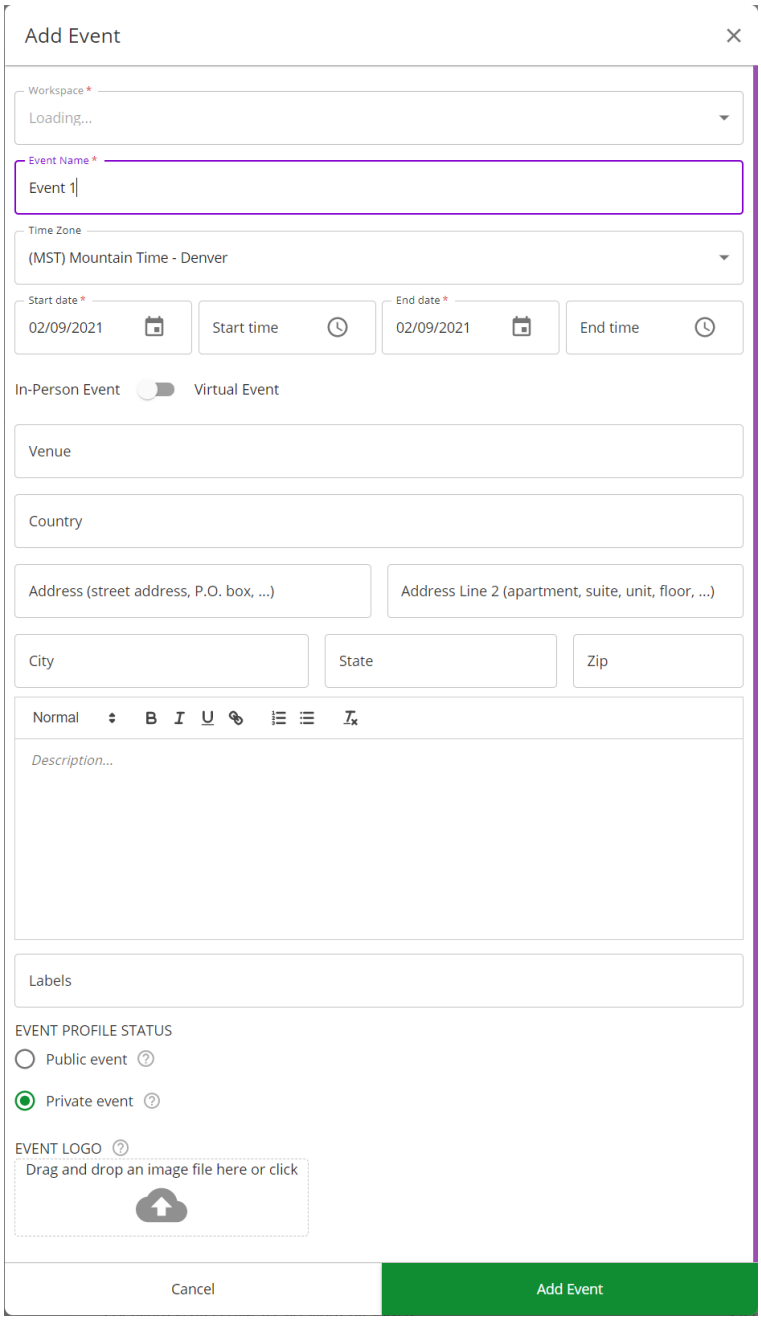
Zvolený postup riešenia zadaných úloh

Zadané úlohy som riešil postupne v poradí, v akom mi boli zadané. Ak som naraził na nejaký problém pri práci na úlohách, obrátil som sa na kolegov, ktorí mi problém viac objasnili, prípadne mi pomohli ho vyriešiť. Pri riešení úloh som často konzultoval moju prácu s kolegami, aby som sa uistil, či úlohy implementujem správne a či sa nedá moja implementácia zefektívniť.

4.1 Formulár vytvorenia udalosti

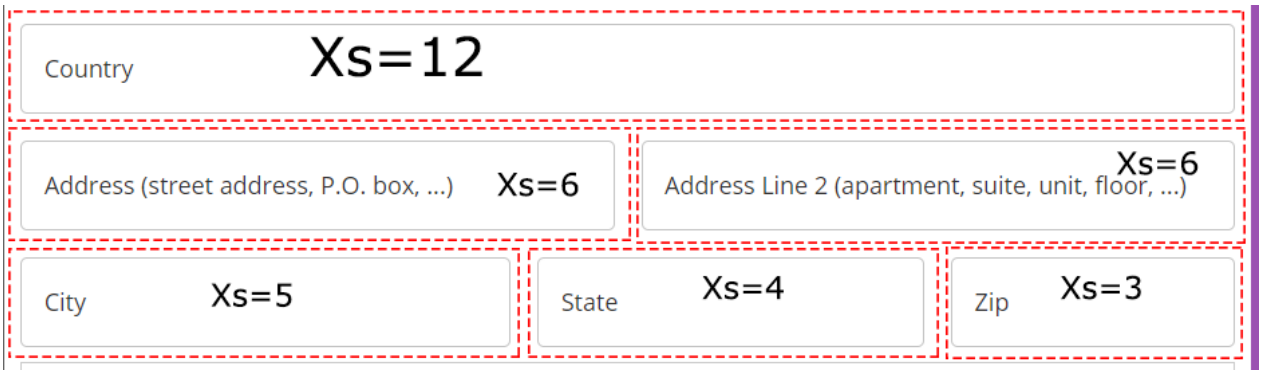
Formulár bude poskytovať používateľovi intuitívny proces vytvárania novej udalosti. V predchádzajúcej verzii formulára bol proces vytvárania udalosti nedopracovaný a používal zastaralý dizajn, a preto bolo tento formulár treba implementovať na novo. Formulár bude implementovaný ako samostatný komponent, ktorý sa bude zobrazovať v modálnom okne. Toto modálne okno sa používateľovi otvorí po kliknutí na tlačidlo vytvorenia nového eventu. Formulár je zložený z niekoľkých menších komponentov určených na používateľský vstup. Používateľ bude do týchto polí zadávať informácie o novom evente.

Podľa grafického návrhu 4.1 som si rozložil formulár na komponenty. Komponenty vyžadujúce zložitejšiu funkcionalitu boli pre mňa vopred pripravené. Jedná sa napríklad o komponent na výber workspace, ktorý na pozadí načíta zoznam používateľových workspacov a dá používateľovi možnosť vybrať jeden z nich. Ku jednotlivým vstupným komponentom bolí definované limity, ako napríklad maximálna veľkosť súboru pri nahrávaní loga alebo maximálna dĺžka textu v textových vstupoch. Okrem limitov vstupov bola definovaná aj povinnosť jednotlivých polí. Ak používateľ nevyplní všetky povinné polia, nebude mu umožnené kliknúť na tlačidlo *Add Event*.



Obr. 4.1: Grafický návrh formuláru udalosti

Aby boli komponenty správne usporiadané a vyzerali tak, ako na grafickom návrhu, vložil som ich do komponentu Grid z Material UI. Grid je formou responzivného kontajneru, ktorý umožňuje usporiadať vnútorné komponenty do stĺpcov a riadkov. Každému stĺpcu sa určí jeho veľkosť (*parameter xs*) relatívne ku riadku, v ktorom je umiestnený. Maximálna šírka stĺpca je 12. Ak nastavíme stĺpcu šírku 4, bude zaberať iba jednu tretinu riadku. Rozloženie je možné vidieť na obrázku 4.2.

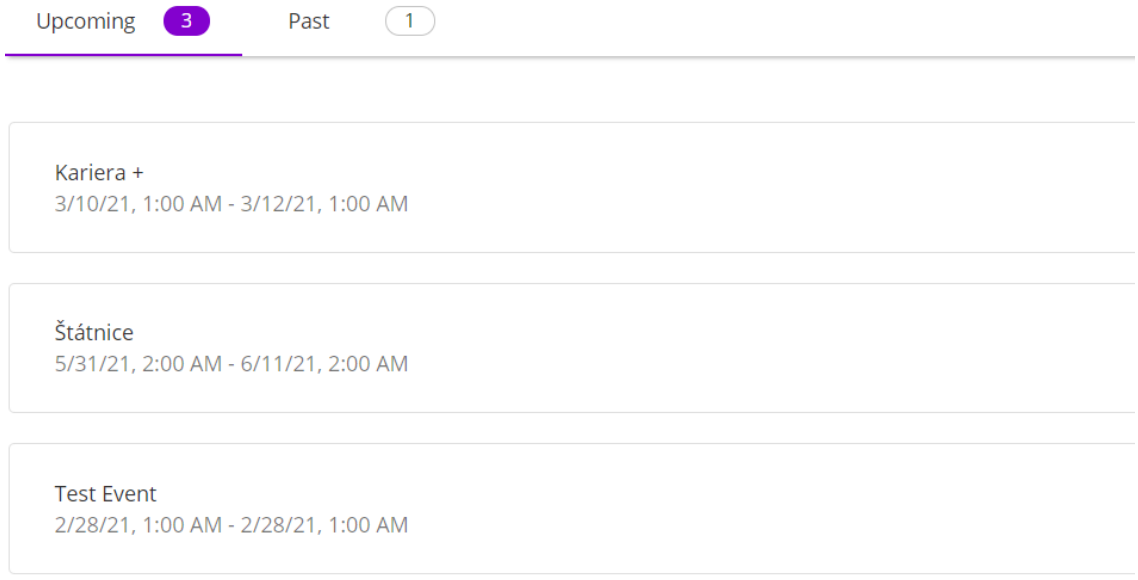


Obr. 4.2: Rozloženie komponentov v Grid

Po rozložení a pridaní komponentov na používateľské vstupy do gridu, som implementoval funkciu na spracovanie dát formuláru, ktorá bude volaná pri uložení formuláru a má za úlohu zavolanie GraphQL dotazu na vytvorenie udalosti. Funkcia postupne plní dátový objekt novo vytvoreného podujatia údajmi, ktoré používateľ vyplnil do formuláru. Dátový objekt sa po naplnení dátami pošle GraphQL dotazom na backend, kde sa dáta spracujú a vytvorí sa v databáze nový event. Dotaz po úspešnom dokončení vráti dátový objekt nového eventu z databázy. Z vráteného dátového objektu následne funkcia vezme ID zložky príloh pre dané podujatie a toto ID spolu s nahratým logom eventu pošle ďalším GraphQL dotazom na backend. Používateľom nahrané logo sa uloží do zložky príloh eventu a dotaz vráti ID súboru - loga, podľa ktorého sa dá logo následne identifikovať. Na koniec funkcia ešte zavolá GraphQL dotaz na aktualizáciu údajov podujatia, pomocou ktorého sa pridá ID loga k podujatiu. Ak všetky kroky prebehli úspešne, modálny formulár sa zatvorí a presmeruje používateľa na hlavnú nástenku. Ak došlo k nejakej chybe pri vytváraní, zobrazí sa chybová hláška.

Majitelia platformy sa neskôr rozhodli, že počet podujatí, ktoré môže používateľ vytvoriť pod jedným workspace, bude limitovaný úrovňou predplatného. Bolo tak treba upraviť funkcionalitu formuláru, aby nedovolil používateľovi vytvoriť event, ak už dosiahol limit eventov. Do funkcionality formuláru som teda musel pridať ďalší GraphQL dotaz, ktorý získa počet podujatí vybraného workspace, spolu s úrovňou predplatného. Ak je počet podujatí vo workspace pod limitom, tak je umožnené používateľovi vytvoriť nové podujatie. Ak je však limit už dosiahnutý, formulár sa zablokuje a zobrazí sa hláška, ktorá informuje používateľa o tom, že už dosiahol maximálny počet podujatí a ak ich chce viac, musí si zakúpiť vyššiu úroveň predplatného.

Po dokončení implementácie som overil, že sa dizajn formuláru zhoduje s grafickým návrhom a otestoval som, či formulár a validácia vstupných polí funguje správne. Skúsil som vytvoriť nový event a po chvíli spracovávania bolo evidentné, že event sa úspešne vytvoril a web stránka bola presmerovaná na nástenku workspacu. Novo vytvorené podujatie som mohol nájsť v zozname podujatí na nástenke, ktorú možno vidieť na obrázku 4.3.



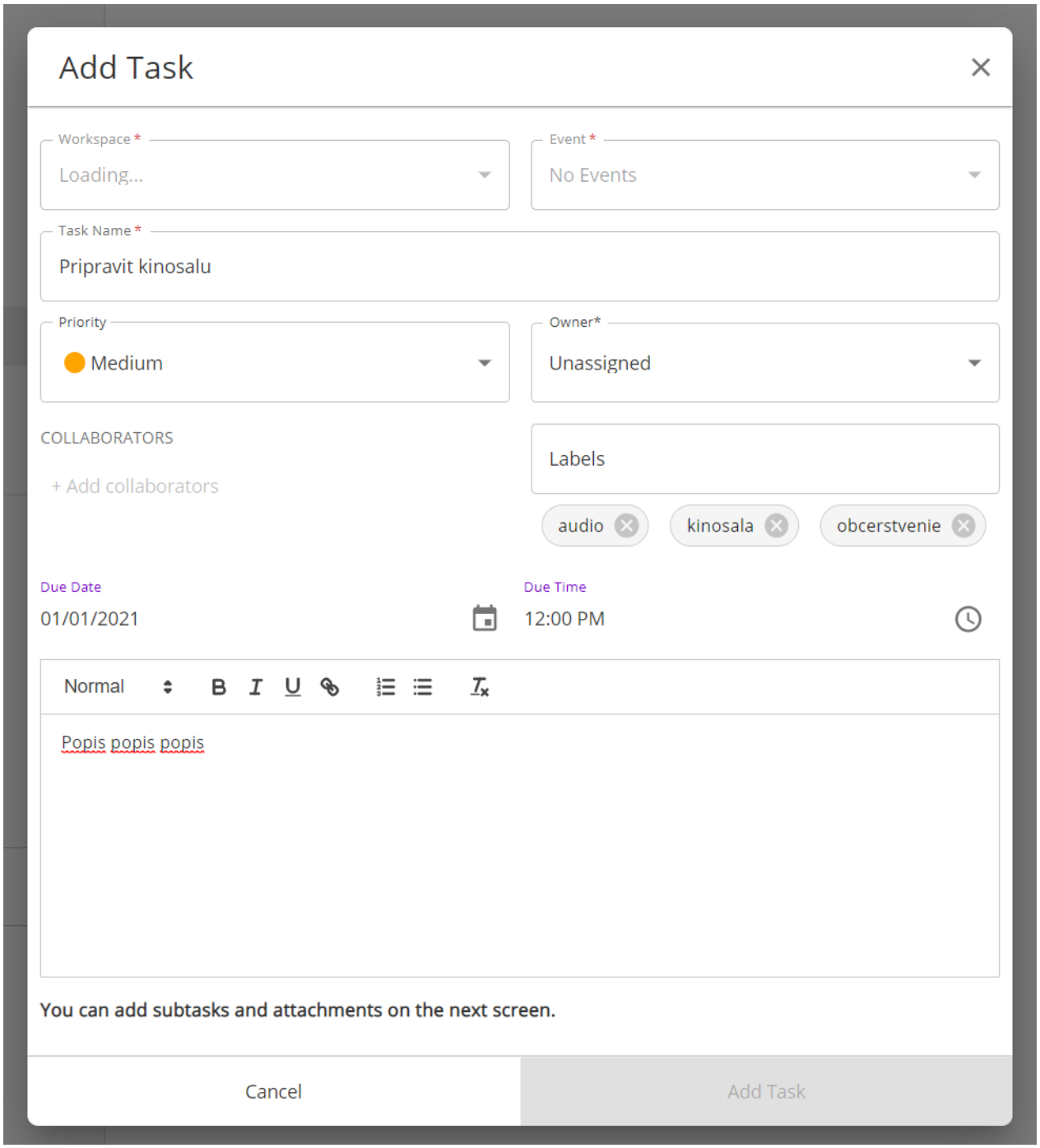
Obr. 4.3: Zoznam podujatí

4.2 Formulár vytvorenia úlohy k udalosti

Tento formulár poskytne používateľovi možnosť pridať ku podujatiam rôzne úlohy. Jedná sa o novú funkcionalitu v aplikácií, a tak ho treba naimplementovať ako nový komponent. Komponent formuláru bude, rovnako ako v predchádzajúcom formulári, zobrazovaný v modálnom okne po tom, čo používateľ klikne na tlačidlo pridania úlohy.

Formulár sa skladá z niekoľkých komponentov určených na používateľský vstup. Ich rozmiestnenie bolo dané grafickým návrhom 4.4. Komponentom určeným na používateľský vstup bola definovaná povinnosť vyplnenia a limity na maximálny počet znakov.

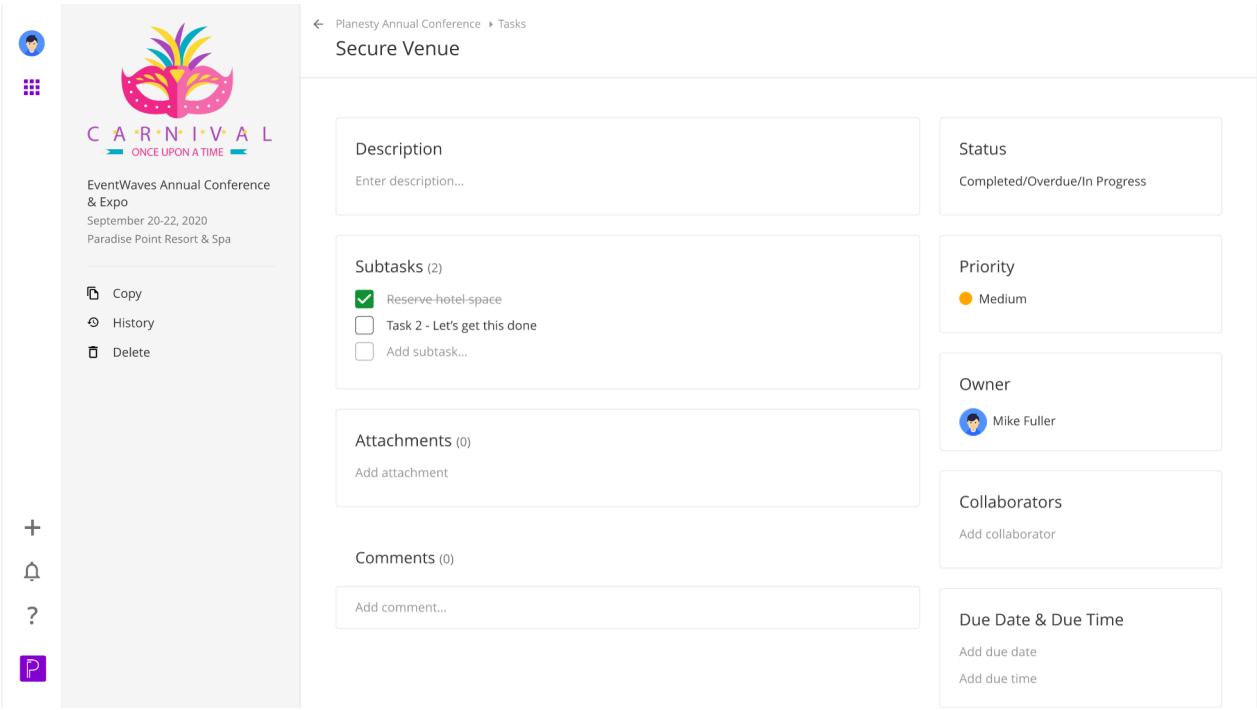
Na správne usporiadanie komponentov som použil komponent Grid. Po rozmiestnení komponentov do gridu som vytvoril funkciu, ktorá sa volá po dokončení formuláru v prípade ak boli vyplnené všetky povinné vstupy formulára. Funkcia zavolá GraphQL dotaz na vytvorenie úlohy a do dotazu sa pošlú používateľom vyplnené dáta z formuláru. Následne program počká na dokončenie dotazu. Po úspešnom vytvorení úlohy sa modálne okno s formulárom zatvorí a presmeruje používateľa na detail danej úlohy. Ak však dôjde ku chybe počas volania GraphQL dotazu, zobrazí sa používateľovi chybová hláška.



Obr. 4.4: Grafický návrh formuláru úloh

Svoju implementáciu som následne otestoval tak, že som skúsil vytvoriť novú úlohu ku náhodnému podujatiu. Po vyplnení povinných aj nepovinných vstupov vo formulári a kliknutí na tlačidlo *Add Task*, bola web stránka presmerovaná na detail novej úlohy, ktorý je možné vidieť na obrázku 4.5.

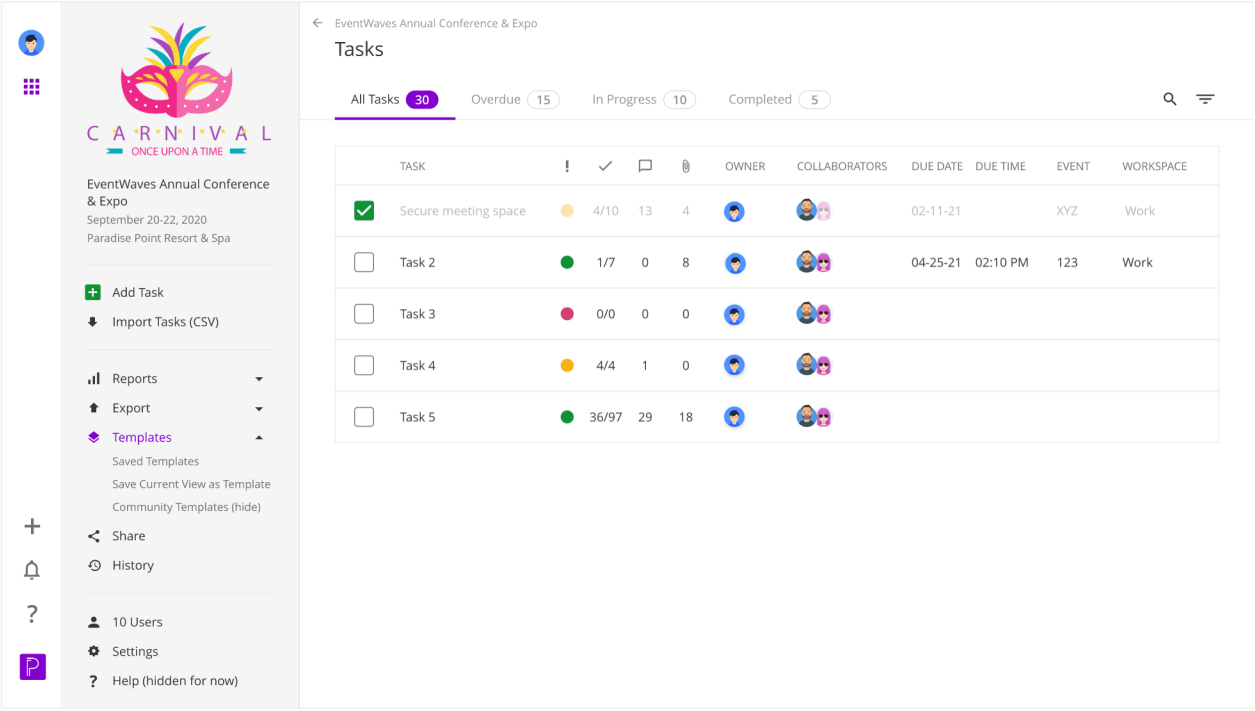
To znamenalo, že funkcia sa správne vykonala a dokončila. Následne som overil, či sa správne uložili všetky informácie o úlohe, a či sa správne zobrazujú na detaile úlohy.



Obr. 4.5: Detail úlohy

Pri testovaní som však objavil chybu pri validácií dĺžky textových reťazcov, zadávaných do formuláru. Dĺžku textového reťazca zo vstupného pola som pôvodne overoval cez vstavanú funkciu *string.length*, táto funkcia však počíta počet unicode [11] znakov v texte. Tu nastáva problém v situácií, keby používateľ zadal do textového pola znaky, ktoré sú zložené z viacerých unicode znakov, no predstavujú graficky len jeden viditeľný znak. Ide napríklad o slová zložené z čínskych znakov alebo emotikony. Pri limite 20 znakov na názov eventu by sa dalo týchto 20 znakov vyčerpať už piatimi emotikonmy. Ide o veľmi špecifický scenár, ku ktorému s veľkou pravdepodobnosťou nikdy nedôjde, no zo zaujímavosti a výzvy som sa rozhodol skúsiť tento problém vyriešiť. Ako riešenie mi napadlo rozložiť textový reťazec do poľa znakov a následne spočítať počet prvkov v poli. Overil som funkčnosť tohto riešenia a následne som vytvoril spoločnú funkciu, ktorá zo vstupného textového reťazca zistí počet viditeľných znakov. Touto funkciou som nahradil funkciu *string.length* pri validácií dĺžky textových reťazcov. Funkcia sa neskôr použila v ostatných komponentoch aplikácie, kde bolo treba overovať dĺžku textových vstupov.

Znova som skúsil vytvoriť novú úlohu a tentokrát už obmedzenie dĺžky textu fungovalo správne. Novú úlohu je možné vidieť aj v zozname úloh vybraného podujatia na obrázku 4.6 alebo v zozname úloh na hlavnej nástenke. Na hlavnej nástenke sa zobrazujú úlohy zo všetkých podujatí daného workspace.



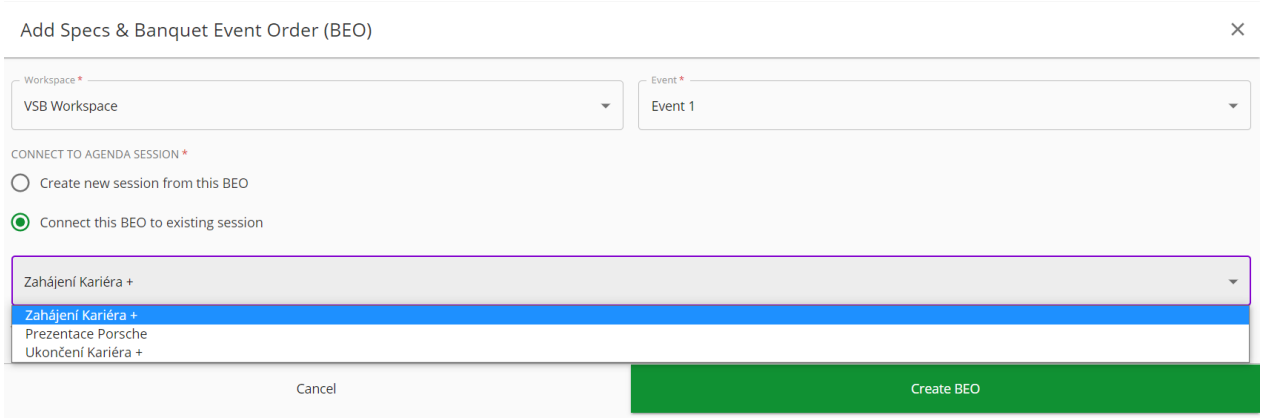
Obr. 4.6: Zoznam úloh vybraného podujatia

4.3 Formulár vytvorenia objednávky služieb - BEO

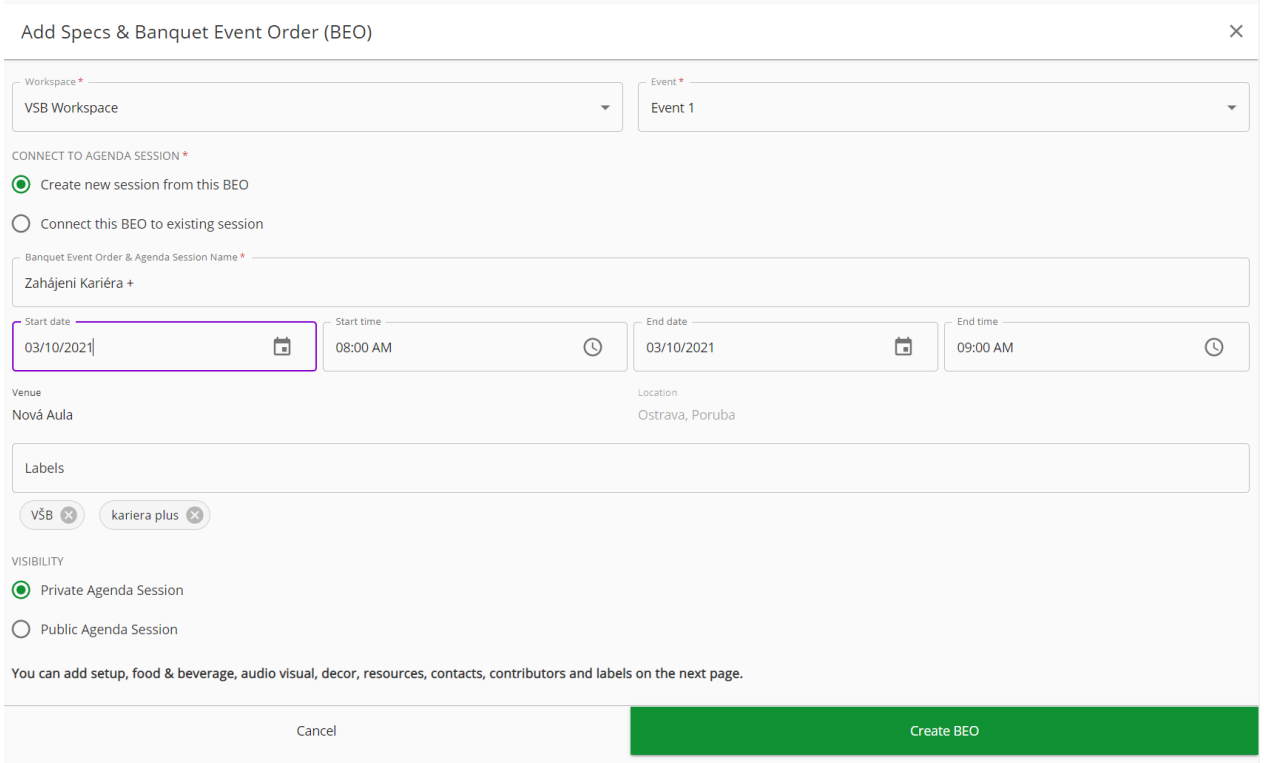
Formulár prinesie do aplikácie používateľovi možnosť vytvorenia objednávky služieb ku podujatiu.

Prácu na formulári vytvorenia objednávky služieb som začal tým, že mi kolegovia vysvetlili, ako funguje BEO a čo to vlastne je, a že každé BEO je priradené k nejakému zhromaždeniu - session alebo activity. V objednávke služieb si používateľ stanovuje počet a typ jedál, nápojov, a audiovizuálnych pomôcok, ktoré majú byť vopred pripravené na dané zhromaždenie. Pri vytváraní BEO si používateľ môže vybrať, či chce novo vytvorené BEO priradiť k nejakej už existujúcej session alebo chce vytvoriť novú. Ak si užívateľ vyberie, že chce vybrať už existujúcu session, zobrazí sa mu zoznam zhromaždení daného podujatia, z ktorého si môže vybrať zhromaždenie. Ak si však užívateľ vyberie, že chce vytvoriť novú session k novému BEO, vo formulári sa mu zobrazia dodatočné informácie, ktoré musí vyplniť. Funkcionalita pridávania jedál, nápojov a audiovizuálnych pomôcok bola ale nakoniec presunutá z formuláru vytvárania BEO do detailu BEO, a tak implementácia tejto funkcionality nebude súčasťou tejto práce.

Grafické návrhy pre tento formulár boli dva: 4.7 a 4.8. Prvý bol grafický návrh formuláru v prípade ak si používateľ vybral možnosť pripojiť BEO k už existujúcej aktivite. Druhý grafický návrh bol rozšírením prvého o dodatočné prvky potrebné na vytvorenie novej aktivity. Podľa grafických návrhov som si spracoval rozmiestnenie komponentov v gride.



Obr. 4.7: BEO formulár s výberom session



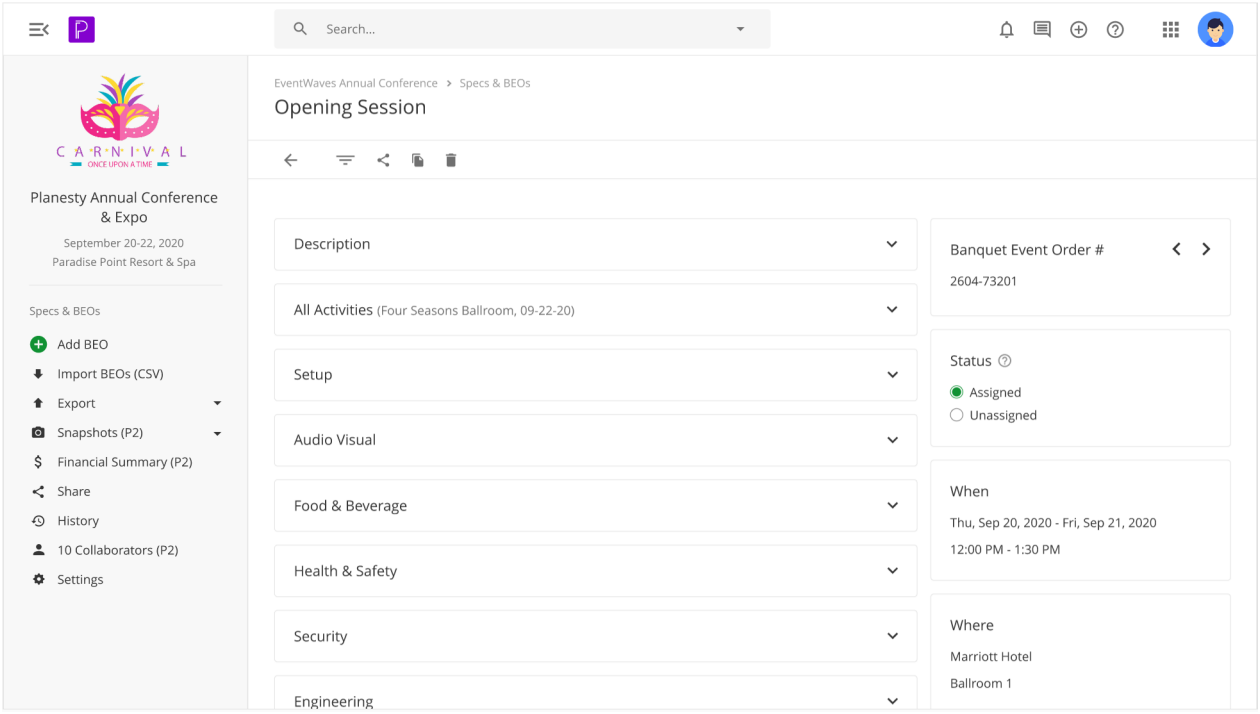
Obr. 4.8: BEO formulár s vytvorením novej session

Ako prvé som sa rozhodol vyriešiť rozdielny vzhľad formuláru podľa vybranej možnosti. Po preskúmaní možností, ako sa dá vyriešiť dvojitý vzhľad jedného komponentu som sa rozhodol použiť podmienené vykresľovanie, ktoré umožňuje vykresľovať rôzny obsah na základe nejakej podmienky. Svoju voľbu som konzultoval s kolegami a tí mi potvrdili, že v tomto prípade je to najvýhodnejšia možnosť riešenia dvojitého vzhľadu. V komponente som preto vytvoril premennú, ktorej hodnota sa bude meniť na základe toho, či používateľ vyberie možnosť, že chce pripojiť BEO ku existujúcej

session alebo možnosť vytvorenia novej session.

Následne som rozmiestnil všetky komponenty používateľského vstupu do vykresľovacej časti komponentu a časť vstupov som obalil podmieneným vykresľovaním, aby sa zobrazovali iba v prípade, ak chce používateľ vytvoriť novú session.

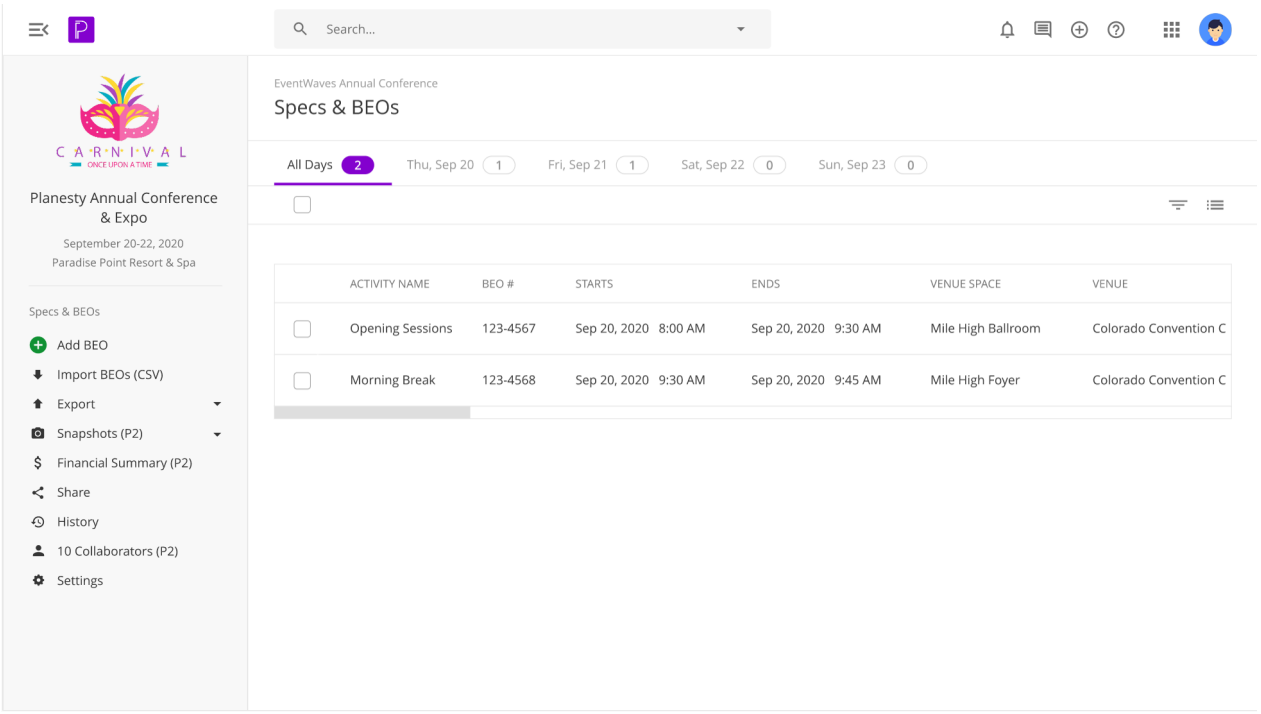
Ďalším krokom bolo vytvoriť funkciu spracovania formuláru. Ako prvé si funkcia poskladá dátový objekt nového BEO a dáta získa z vyplneného formuláru. Ak chce používateľ vytvoriť novú session, zavolá sa GraphQL dotaz, ktorý vytvorí novu session. Funkcia počká na skončenie dotazu, ktorý po dokončení vráti ID novo vytvorenej session. Toto ID sa pridá do dátového objektu BEO. Ak chce používateľ pripojiť BEO k už existujúcej session tak sa ID session, ktorú vybral vo formulári pridá do dátového objektu nového BEO. Následne sa zavolá ďalší GraphQL dotaz na vytváranie BEO s dátovým objektom nového BEO na vstupe. Keď sa dotaz dokončí, tak vráti ID novo vytvoreného BEO a pokiaľ nedošlo k chybe pri volaní dotazov, tak sa formulár zavrie, a presmeruje používateľa na detail nového BEO, ktorý je možné vidieť na obrázku 4.9.



Obr. 4.9: Detail BEO

Pri vytváraní nových session bolo dôležité zabezpečiť, aby sa rôzne session, ktoré sú na rovnakom mieste konania, časovo neprekrývali. Ak sa prekrývali, tak GraphQL dotaz, ktorý mal za úlohu vytvoriť novú session, vrátil errorovú správu o kolízii session. Túto chybu bolo treba odchytávať vo formulári a v prípade, že ku tomuto erroru došlo, zablokovať tlačidlo dokončenia formuláru, a zobraziť používateľovi správu, informujúcu o prekrytí dvoch session, a potrebe zmeniť časový rozsah alebo miesto konania. Po tom čo používateľ upraví miesto konania alebo časový rozsah tak sa znovu odomkne tlačidlo dokončenia formuláru.

Po dokončení všetkej funkcionality som overil správnu funkčnosť formuláru. Vyskúšal som vytvoriť novú objednávku ku podujatiu a po spracovaní bola web stránka presmerovaná na detail novej objednávky. Novú objednávku som mohol následne nájsť aj v zozname objednávok celého podujatia 4.10.



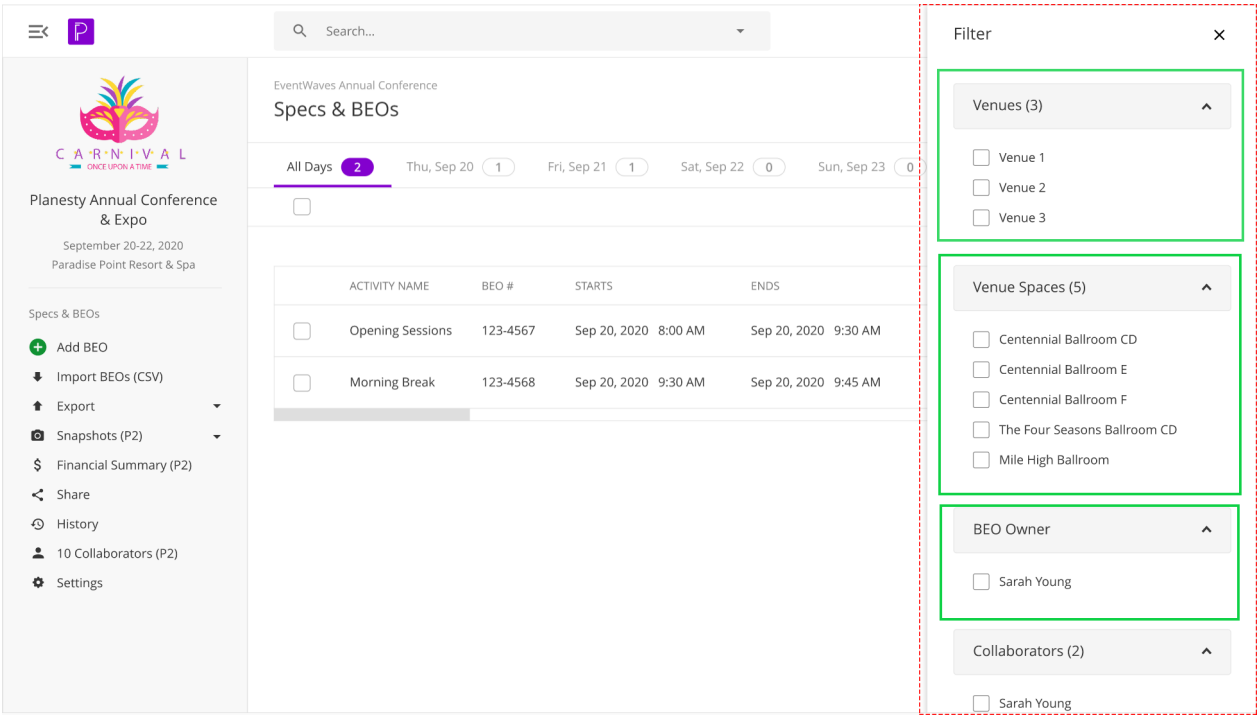
Obr. 4.10: Zoznam objednávok

4.4 Filtrovacie menu objednávok

Ďalšou mojou úlohou bolo implementovať filtrovacie menu, ktoré umožní používateľovi filtrovať zo zoznamu objednávok. Komponent filtračného menu bude vložený do grafického vysúvacieho komponentu. Filtračné menu bude schované a bude sa vysúvať z pravej strany užívateľského rozhrania po tom, čo používateľ klikne na tlačidlo. Filtrovacie menu je rozdelené podľa parametrov BEO, každý parameter je reprezentovaný ako zoznam hodnôt, ktorý sa dá otvoriť alebo zatvoriť pre lepšiu pre-

hľadnosť. V zozname hodnôt sa zobrazujú všetky hodnoty, ktoré sa pri danom parametre vyskytujú vo všetkých BEO z listu. Používateľ si tak môže vybrať hodnoty pre každý parameter a následne sa mu zo zoznamu BEO vyfiltrujú len tie BEO, ktoré spĺňajú hodnoty zadané vo filtrovaciom menu.

Ako prvé som zvolil, podľa ktorých parametrov objednávok by malo zmysel zoznam filtrovať. Po konzultácií s kolegami a majiteľmi platformy sme sa zhodli na tomto zozname parametrov, podľa ktorých sa bude dať filtrovať a následne sme počkali, kým nám pripravia grafický dizajn zodpovedajúci vybraným parametrom. Pripravený graficky dizajn je možné vidieť na obrázku 4.11.



Obr. 4.11: Otvorené filtrovacie menu

Komponent menu je zložený z niekoľkých rovnakých sub-komponentov, ktoré obsahujú zoznam hodnôt určitého parametru. Tieto sub-komponenty je možné vidieť zvýraznené zelenou farbou na obrázku 4.11. Z tohto dôvodu som si vytvoril sub-komponent, do ktorého bude vstupovať názov parametru, zoznam jeho hodnôt a zoznam odkliknutých hodnôt. Sub komponent sa dá rozrolovať alebo zarolovať a obsahuje funkciu, ktorá zavolá predanú callback funkciu pri kliknutí na nejakú z hodnôt.

Do komponentu filtrovacieho menu vstupuje zoznam filtrov, ktoré reprezentujú parametre a ich zoznamy hodnôt, zoznam používateľom odkliknutých hodnôt, a callback funkcia, ktorá bude volaná pri zmene vybranej hodnoty.

Komponent pomocou cyklu prejde všetkými prvkami zo zoznamu filtrov a pre každý filter pridá

sub-komponent daného filtra do render funkcie a predá mu dáta filtru. Výsledkom bude N[1] otváracio/zatváracích komponentov vo finálnom filtrovaciom menu. Vďaka tomuto prístupu nepredstavuje škálovateľnosť filtrovacieho menu žiaden problém. Keď sa majitelia platformy rozhodnú, že chcú pridať ďalšie parametre filtrovania, stačí tieto parametre pridať do zoznamu filtrov a filtrovacie menu sa samo o tieto parametre rozšíri.

Po dokončení komponentu filtrovacieho menu som tento komponent vložil do komponentu vysúvacieho menu a tento komponent som pridal do komponentu zobrazujúceho zoznam BEO. Vytvoril som premennú, ktorá určovala viditeľnosť menu a pridal tlačidlo, po ktorého kliknutí sa zmení stav premennej, a tým sa vysunie filtrovacie menu. Komponent vysúvacieho menu obsahuje tlačidlo na zatvorenie menu, ktoré volá predanú funkciu. Do komponentu vysúvacieho menu som teda predal funkciu, ktorá zmení hodnotu premennej viditeľnosti menu po kliknutí na tlačidlo zavrieť. Ďalej som vytvoril dátový objekt obsahujúci dohodnuté filtrovacie parametre s prázdnym zoznamom možných hodnôt, prázdnym zoznamom vybraných hodnôt, názvami parametrov a funkcií na aktualizáciu výberu hodnôt. Tento dátový objekt som predal do komponentu filtrovacieho menu.

Predtým ako sa dátový objekt predá komponentu filtrovacieho menu však treba naplniť zoznam možných hodnôt. K tomuto účelu som vytvoril funkciu, ktorá načíta cez GraphQL dotaz zoznam všetkých objednávok. Následne funkcia prejde cyklom všetky objednávky a ukladá všetky hodnoty, ktoré sa pod parametrami z filtrov vyskytujú v objednávkach. Po dokončení cyklu sa zoznamy možných hodnôt pridajú k daným parametrom do dátového objektu filtrov.

Filtrovacie menu funguje tak, že keď používateľ zaklikne nejakú hodnotu z nejakého parametru, zavolá sa callback funkcia, ktorá aktualizuje vybrané hodnoty daného parametru v dátovom objekte filtrov. Keď dôjde k nejakej zmene vo vybraných hodnotách filtrov, tak sa spustí funkcia na prefiltrovanie zoznamu BEO. Ďalšou úlohou bolo implementovať túto funkciu.

Funkcia na filtrovanie zoznamu BEO načíta zoznam všetkých BEO aktuálne používaného eventu pomocou GraphQL dotazu. V cykle prechádza všetkými týmito BEO a porovnáva, či sa parametre BEO zhodujú s tými, ktoré sú nastavené vo filtri. Konkrétne s tými, ktoré sú v položke používateľom zakliknutých hodnôt. Ak sa zhodujú, tak takéto BEO, ktoré spĺňa filtrovacie kritéria, pridá do zoznamu vyfiltrovaných objednávok. Po dokončení cyklu, funkcia vráti vyfiltrované objednávky a tie sa následne zobrazia používateľovi.

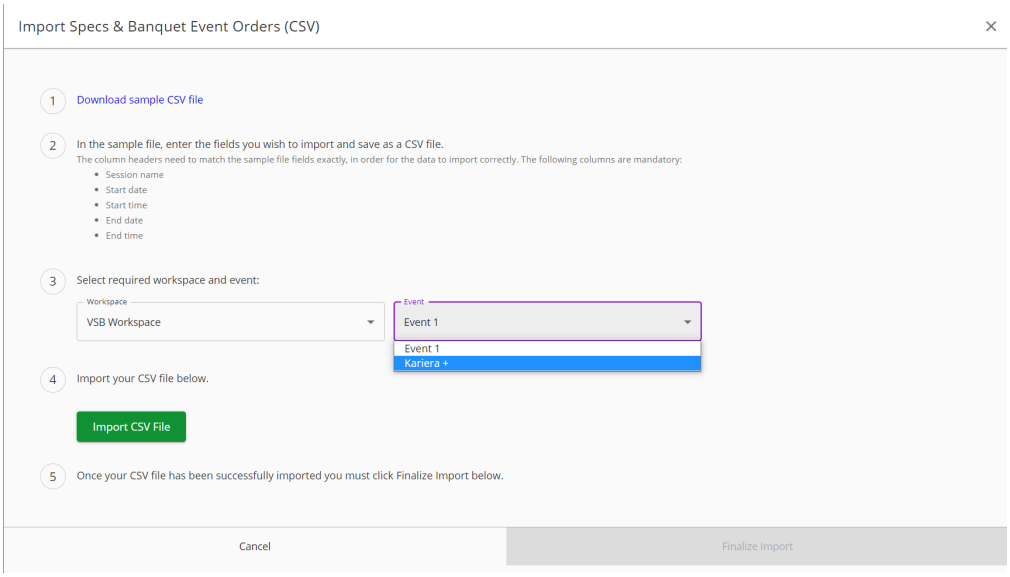
Keď som dokončil implementáciu komponentu, otestoval som, či filtrovanie funguje správne a či filtrovanie netrvá príliš dlho. Objednávky sa po zadaní filtrov vyfiltrovali skoro okamžite a správne, takže filtrovacie menu funguje korektne.

[1]N = počet položiek v zozname filtrov

4.5 Menu na import CSV súboru

Menu bude slúžiť na nahranie CSV súboru, v ktorom je zoznam objednávok. Je to alternatívny spôsob, ako vytvárať objednávky k podujatiu a pri väčšom množstve objednávok je toto pre používateľa výhodnejšia a rýchlejšia možnosť. Menu sa skladá z informačných komponentov, ktoré stanovujú presný formát názvov stĺpcov v CSV súbore, ktorý musí používateľ dodržať, aby sa dali dáta importovať. V ďalšej časti menu je možnosť vybrať, pod ktorým workspace a pre ktoré podujatie sa majú objednávky vytvoriť. Po tom, čo používateľ vyberie workspace a podujatie, môže nahrať CSV súbor a potvrdiť importovanie.

Implementáciu som začal rozložením komponentov podľa grafického návrhu 4.12. Do komponentu som pridal podmienky, ktoré obmedzovali kliknutie na tlačidlo dokončenia importu pokiaľ používateľ nemá vybratý workspace, event a nahraný CSV súbor. Po potvrdení komponent importu pridá ku súboru ID podujatia, ID workspace a dáta pošle cez GraphQL dotaz na backend, kde sa vykoná validácia CSV súboru. Ak nedôjde k chybe a dáta sú validné, dáta sa spracujú a vytvoria sa podľa nich objednávky služieb ku danému podujatiu. Program počká na dokončenie spracovávania dát na backende a po správnom dokončení zobrazí informačnú hlášku o úspešnom spracovaní CSV súboru. Následne používateľa presmeruje na zoznam objednávok vybraného podujatia. Ak dôjde pri spracovávaní dát na backende k chybe, backend túto chybu vráti a zobrazí sa používateľovi. Je dôležité, aby používateľ dodržal presne špecifikované názvy stĺpcov v CSV súbore. Ak tieto názvy nedodrží tak sa z backendu vráti chyba o nesprávnych názvoch stĺpcov a táto chyba sa zobrazí vo formulári.

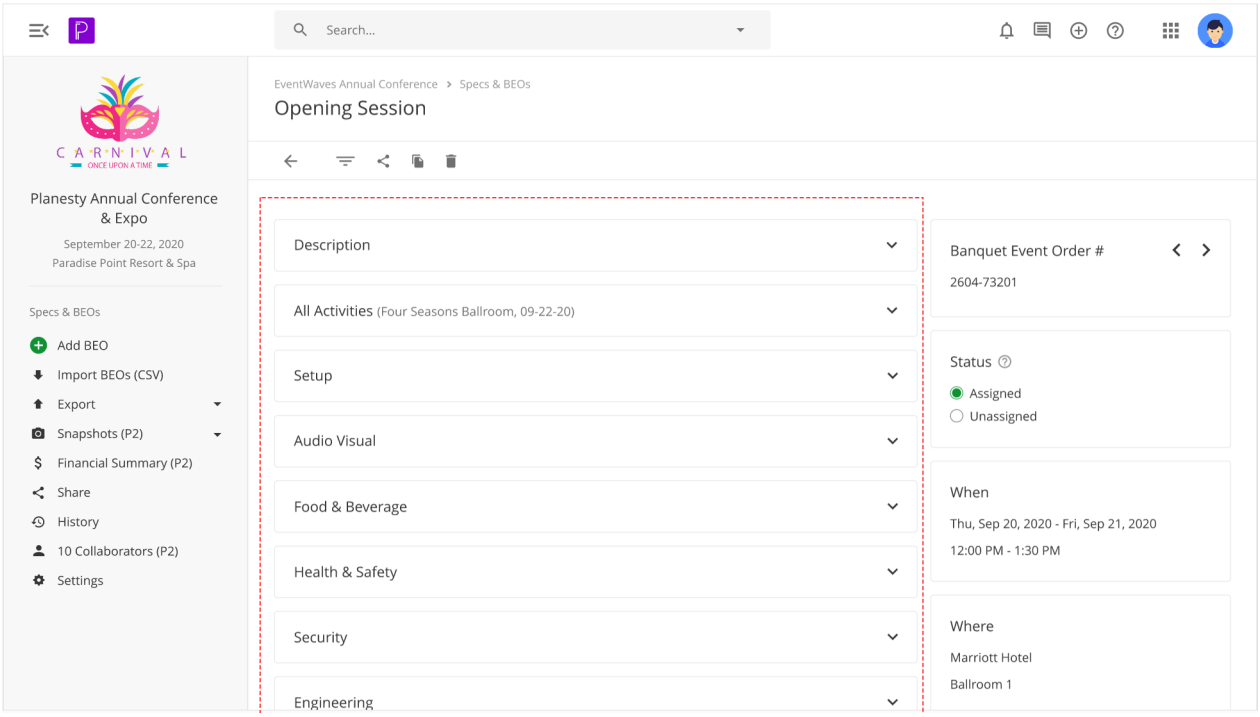


Obr. 4.12: Menu CSV importu

Po implementácii som vyskúšal cez menu nahrať validný CSV súbor a po spracovaní som videl pribudnúť nové objednávky do zoznamu objednávok. Potom som vyskúšal nahrať aj nevalidný súbor a menu zobrazilo chybovú hlášku, čo znamenalo správnu funkčnosť importovacieho menu.

4.6 Editácia viditeľnosti informačných prvkov

Mojou poslednou úlohou bolo pridať menu do detailu BEO, cez ktoré si používateľ môže povoliť viditeľnosť informačných prvkov BEO. Ako také informačné prvky vyzerajú, je možné vidieť na obrázku 4.13. Používateľ si tak môže zmeniť, čo sa mu zobrazuje v grafickom rozhraní a prispôsobiť si ho podľa svojich potrieb. Menu sa bude vysúvať z bočnej strany web stránky, rovnako ako pri filtrovaciom menu. Menu sa skladá zo zoznamu názvu informačných prvkov a zaškrtávacieho pola vedľa názvu.



Obr. 4.13: Detail BEO s informačnými prvkami

Pre menu som vytvoril nový komponent a pridal doň cyklus, ktorý bude prechádzať zoznamom názvov, ktoré budú do komponentu predané z rodičovského komponentu. Okrem zoznamu názvov, vstupuje do komponentu aj zoznam názvov, ktoré používateľ zaškrtol a callback funkcia, ktorá bude volaná pri zmene zaškrtnutia nejakého políčka.

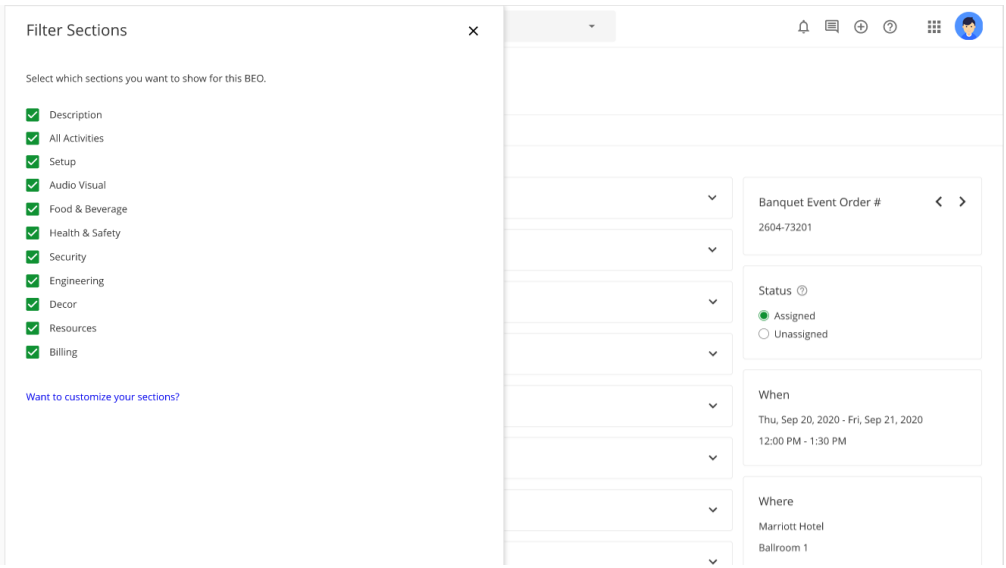
Pre každý názov prvku sa do komponentu pridá sub-komponent obsahujúci názov prvku a zaškrtávacie políčko vedľa neho. Ak sa nachádza daný prvok aj v zozname zaškrtnutých polí, zaškrtávacie políčko sa nastaví ako zaškrtnuté.

Ku zaškrtávaciemu políčku sa pridá funkcia, ktorá sa zavolá pri zaškrtnutí alebo odškrtnutí. Keď používateľ zaškrtne alebo odškrtne nejaký informačný prvok, zavolá sa callback funkcia rodičovského komponentu a vstupným parametrom tejto funkcie je názov zaškrtnutého prvku. Funkcia v rodičovi následne skontroluje či v zozname zaškrtnutých názvov tento názov už nie je, a ak je tak ho odtiaľ odstráni. Ak tam však nie je tak ho do zoznamu pridá. Zoznam zaškrtnutých polí sa aktualizuje a tým sa v komponente menu zmení stav zaškrtnutia, a v detaile BEO sa upraví viditeľnosť aktualizovaných prvkov.

Hotový komponent menu som následne pridal do komponentu zobrazujúceho detail BEO a vytvoril v ňom zoznam prvkov a zoznam zaškrtnutých prvkov. Ďalej som vytvoril funkciu, ktorá bude pridávať alebo odoberať prvky zo zoznamu zaškrtnutých polí. Tieto zoznamy a funkciu som predal komponentu menu. V komponente BEO detailu bolo treba upraviť vykresľovaciu časť, a to tak, že som tam pridal podmienky, aby sa každý informačný prvok vykresľoval iba vtedy, ak je v zozname zaškrtnutých prvkov.

Aby používateľ nemusel nastavovať viditeľnosť prvkov zakaždým, keď načíta stránku, bolo potreba pridať na začiatok funkcie GraphQL dotaz, ktorý získa nastavenia daného používateľa. Ak existuje nejaké uložené nastavenie používateľa, tak sa použije a ak neexistuje, tak sa použije predvolený zoznam, v ktorom sú zaškrtnuté všetky polia. Keď používateľ zmení nastavenie zobrazovania, tak sa cez GraphQL dotaz uloží nastavenie a keď potom používateľ navštívi stránku znova, tak sa mu jeho nastavenie načíta.

Po dokončení komponentu som overil, že funguje správne a že sa jeho dizajn zhoduje s grafickým návrhom. Menu je možné vidieť na obrázku 4.14.



Obr. 4.14: Otvorené menu editácie viditeľnosti prvkov

Kapitola 5

Záver

Odborná prax mi priniesla veľa nových skúsenosti a znalosti v oblasti vývoju softwaru. Počas praxe som sa naučil vyvíjať frontend efektívnejšie a na vyššej úrovni. Zdokonalil som sa v programovaní v jazyku JavaScript a naučil sa pracovať s knižnicou ReactJS. Osvojil som si používanie statického typovania v JavaScripte pomocou Typescriptu a túto znalosť som naďalej používal aj vo svojich osobných projektoch.

5.1 Dosiahnuté výsledky v priebehu odbornej praxe a ich celkové zhodnotenie

Výsledkom mojej práce boli funkčné časti používateľského rozhrania používané na spravovanie udalosti. Všetky zadané úlohy sa mi podarilo splniť podľa plánu a s mojou implementáciou boli vo firme spokojní. Niektoré funkcionality, ktoré som vytvoril boli neskôr prepoužité aj na iných miestach. Šlo napríklad o menu CSV importu, ktoré potom na projekte použili okrem importu objednávok aj na import kontaktov alebo sessions.

Po dokončení praxe mi vo firme ponúkli pracovné miesto frontend vývojára, a tak som ostal vo firme naďalej pracovať na platforme Planesty.

Vďaka odbornej praxi som sa zoznámil s procesom vytvárania softvéru a kolaboratívnej práce na spoločnom projekte. Odborná prax mi priniesla skúsenosti ako to v takej firme na vývoj software funguje. Od komunikácie so zákazníkom, návrhu infraštruktúry, rozdeľovania úloh až po implementáciu a testovanie výslednej aplikácie. Naučil som sa, čo sa rozumie pod pojmom agilný vývoj a ako sa pristupuje ku iteratívnemu vývoju.

5.2 Teoretické a praktické znalosti, a zručnosti získané v priebehu štúdia uplatnené študentom v priebehu odbornej praxe

Platforma používa trojvrstvovú architektúru, o ktorej som získal znalosti v predmete Vývoj informačných systémov a vďaka tomu som lepšie porozumel, ako celý systém funguje.

Znalosti o kódovaní a spájaní unicode znakov z predmetu Logických Obvodov sa mi hodili pri analýze problému s nesprávnym počítaním znakov v texte.

So zákazníkom sme komunikovali výhradne v anglickom jazyku, s ktorým mi pomohli hodiny anglického jazyka v prvých štyroch semestroch štúdia.

5.3 Znalosti či zručnosti chýbajúce študentovi v priebehu odbornej praxe

Frontendová časť aplikácie bola programovaná v jazyku JavaScript, s ktorým som mal do tej doby len malé skúsenosti, a tak som sa ho musel najskôr naučiť. JavaScript sme mali v predmete tvorba aplikácií pre mobilné zariadenia avšak tento predmet som mal až v šiestom semestri a do tej doby som sa JavaScript naučil sám.

Na verziovanie zdrojových kódov sa používala technológia GIT [12], s ktorou som sa počas štúdia stretol len veľmi zriedkavo a aj to len so základnými git príkazmi. Chýbali mi znalosti pokročilých git funkcionalít a príkazov ako napríklad rebase alebo cherry-pick. Verziovacie nástroje sú neoddeliteľnou súčasťou vývoju software a používať by ich mal vedieť každý vývojár. Uvítal by som keby sa verziovacím nástrojom venovalo v škole viacej pozornosti.

Pri nasadzovaní aplikácie na server sme vo firme používali CI/CD pipeline, ktorý najskôr spustil nad aplikáciou pripravené testy a ak testami prešla, tak bola nasadená na web server. CI/CD pipeline bol pre mňa úplne neznámym a novým pojmom. Po tom čo mi kolegovia objasnili ako to funguje a troche samoštúdia som pochopil, že je to veľmi nápomocný nástroj pri tvorbe software.

Použitá literatúra

1. *Profiq s.r.o.* [Online] [cit. 2021-03-06]. Dostupné z : `https://www.profiq.com/`.
2. *Planesty* [online] [cit. 2021-03-06]. Dostupné z : `https://www.planesty.com/`.
3. WIERUCH, Robin. *The Road to React: Your journey to master plain yet pragmatic React.js.* Independently published, 2018.
4. CHERNY, Boris. *Programming TypeScript: Making Your JavaScript Applications Scale.* O'Reilly Media, 2019.
5. WIERUCH, Robin. *The Road to GraphQL: Your journey to master pragmatic GraphQL in JavaScript with React.js and Node.js.* Independently published, 2018.
6. *Material UI* [online] [cit. 2021-03-06]. Dostupné z : `https://material-ui.com/`.
7. *CI/CD* [online] [cit. 2021-03-06]. Dostupné z : `https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html`.
8. FLANAGAN, David. *JavaScript: The Definitive Guide.* O'Reilly Media, 2011.
9. *Facebook* [online] [cit. 2021-04-19]. Dostupné z : `https://www.facebook.com/`.
10. *REST API* [online] [cit. 2021-03-06]. Dostupné z : `https://restfulapi.net/`.
11. *Unicode* [online] [cit. 2021-04-19]. Dostupné z : `https://home.unicode.org/`.
12. LOELIGER, Jon. *Version Control with Git: Powerful tools and techniques for collaborative software development.* O'Reilly Media, 2009.